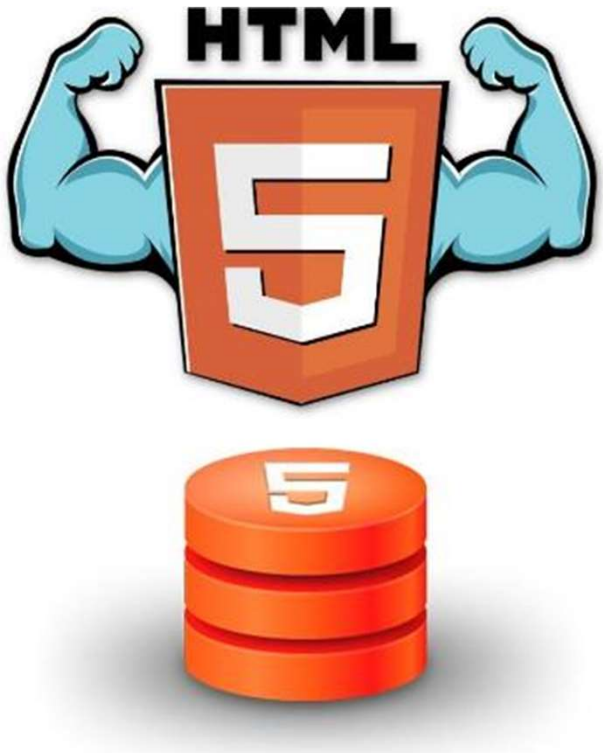
HTML Web Storage

Client-Side Web Programming

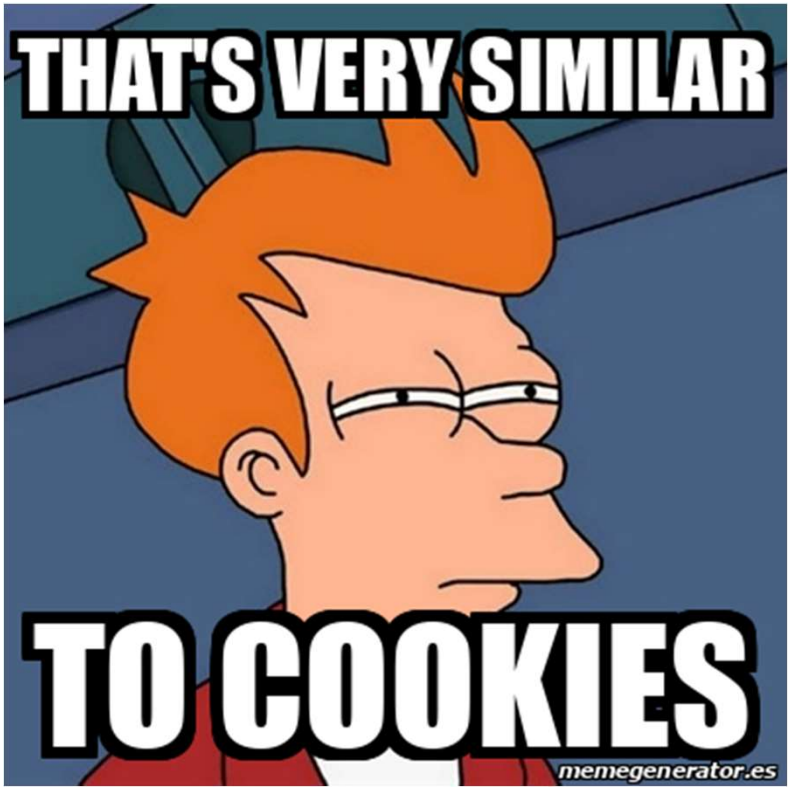
José Socuéllamos

1.- Introducing Web Storage

- HTML5 brought a series of improvements for our webpages.
- One of them is the Web Storage, used to store data locally, on the user computer.
- These data can be used in future sessions.
- But, then…



1.- Introducing Web Storage



2.- Cookies vs Web Storage

COOKIES	WEB STORAGE
Up to 4 KB	Up to 5 MB (Chrome allows larger sizes)
Each time you request something from the server, cookies are sent and received back with the request.	Data is stored IN the browser. It's not sent to the server as a part of the request
Cookies have an expiration date	There is no expiration date for the local web storage, even if we close the browser. If we want the data to be deleted after the browser is closed, we can use session web storage.
Cookies and Web storage are per origin (per domain and protocol). All pages, from one origin, can store and access the same data.	
Web Storage is not more secure than cookies. We must not store sensitive data (like passwords and credit card numbers) on the client-side.	

2.- Local vs Session

- Web Storage always stores data as Strings. Remember to convert it to another format when needed.
  - The localStorage Object
    - It saves key/value pairs in a web browser.
    - It stores the data with no expiration date.
    - The data will not be deleted when the browser is closed, and will be always available.
  - The sessionStorage object
    - It's the same as the localStorage object, except that it stores the data for only one session.
    - The data is deleted when the user closes the browser tab.

3.- How To SAVE Data

- You can save data using the setItem() method or the dot notation (obj.key).

```
var nom = $("inputName").val();
localStorage.setItem("name", nom);
localStorage.name = nom;
```

```
var ape = $("inputSurname").val();
sessionStorage.setItem("surname", ape);
sessionStorage.surname = ape;
```

4.- How To RETRIEVE Data

- You can retrieve data using the getItem() method or the dot notation (obj.key).

```
var storedName = localStorage.getItem("name");
var storedName = localStorage.name;
console.log(storedName);
```

```
var storedSurname = sessionStorage.getItem("surname");
var storedSurname = sessionStorage.surname;
console.log(storedSurname);
```

4.- How To CHECK Data

- You can check if a key already exists by evaluating it in a conditional sentence.

```
if (localStorage.getItem("name")){
    console.log("The name is already saved");
}
else {console.log("The name is not saved yet");}
```

```
if (sessionStorage.getItem("surname")){
    console.log("The surname is already saved");
}
else {console.log("The surname is not saved yet");}
```

5.- How To DELETE Data

- The removeItem() method removes the specified Storage Object item.

```
localStorage.removeItem("name");
sessionStorage.removeItem("surname");
```

- To delete all the data saved in the Web Storage we can use the clear() method:

```
localStorage.clear();
sessionStorage.clear();
```

5.- Storing objects

- Web Storage can only work with Strings, but we can use a workaround to store objects too.
- JSON (JavaScript Object Notation) is a lightweight format for storing and transporting data.
- We can use JSON to transform JavaScript objects into Strings and save them using Web Storage.
- When we need them, we can extract them and convert them back to JS objects.

6.- Storing objects

```
var person = {
    firstName: "James",
    lastName: "Bond",
    job: "spy"
};

localStorage.setItem("user", JSON.stringify(person));
//Some lines later...
var pers = JSON.parse(localStorage.getItem("user"));
console.log(pers.firstName); //James
```

- JSON.stringify(object) - Converts a JavaScript object into a string.
- JSON.parse(string) - Parses the data and converts it into a JS object.

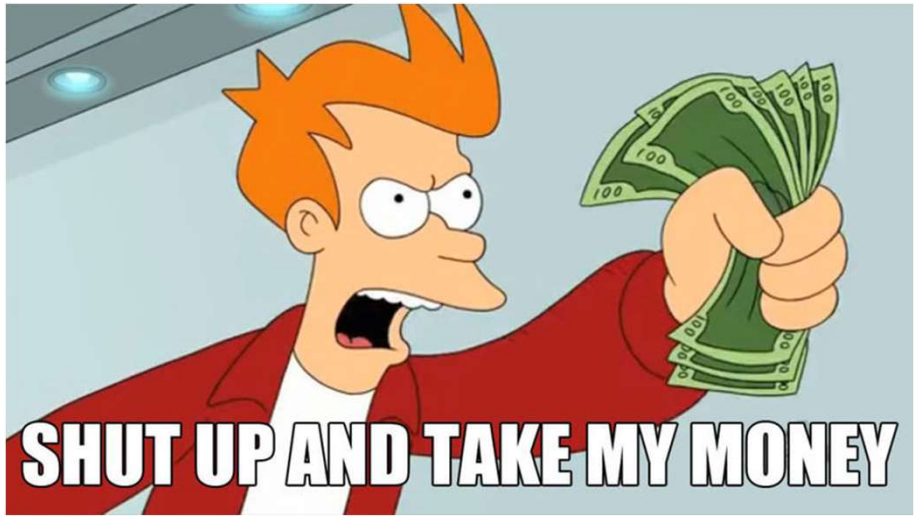
7.- Data Attributes

- Tell me you have never used element class names to store fragments of metadata for the sole purpose of making your JS simpler?
- For instance…
  - We have a list of different restaurants on a webpage.
  - We want to store information about the type of food offered by restaurants or their distance from the visitor.
  - We usually use the HTML class attribute to store that info:

```
<li class="chinese">Li Feng</li>
<li class="pizza">Don Mario</li>
<li class="burger">Meat & Greet</li>
<li class="pizza">Pasta e Pomodoro</li>
```

7.- Data Attributes

- Thanks to HTML5, we now have the ability to embed custom data attributes on all HTML elements.



7.- Data Attributes

- Custom data attributes consists of two parts:
  - Attribute name: must be prefixed with *data-* and has to be at least one character long.
  - Attribute value: can be any String (except for capital letters).

```
<li data-food="chinese" data-id="3781">Li Feng</li>
<li data-food="pizza" data-id="8104">Don Mario</li>
<li data-food="burger" data-id="6382">Meat & Greet</li>
<li data-food="pizza" data-id="9267">Pasta e Pomodoro</li>
```

7.- Data Attributes

- An element can have any number of data attributes with any value you want.

- Be aware that users can see these data so don't add sensitive information in the attribute.

- In the browser, you won't see any difference in the webpage using these attributes.

7.- How can we use Data Attributes?

- Let's see some examples of how to use custom data attributes:
  - To store the initial height or opacity of an element which might be required in later JavaScript animation calculations.
  - To store custom web analytics tagging data.
  - To store data about the health, ammo, or lives of an element in a JavaScript game…
- Data attributes aren't an appropriate solution for all problems:
  - If there is a existing attribute or element which is more appropriate for storing our data (i.e. date/time data).

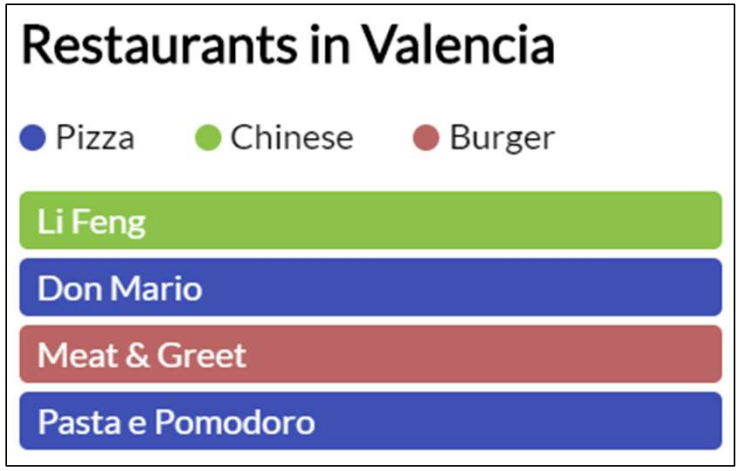
7.- How shouldn't we use Data Attributes?

- Data attributes aren't an appropriate solution for all problems:
  - If there is a existing attribute or element which is more appropriate for storing our data (i.e. date/time data).
  - Marking up contact details or event details using custom data attributes would be wrong, unless it's only intended to be used by our own internal scripts.
  - The presence/absence of a particular data attribute should not be used as a CSS hook for any styling. Doing so would suggest that the data you are storing is of immediate importance to the user and should be marked up in a more semantic and accessible manner.

7.- STYLING Data Attributes

- We can give users a cue about the type of restaurant using attribute selectors to style the restaurants differently.

```
li[data-food='chinese'] {
    background: #8BC34A;
}
li[data-food='pizza'] {
    background: #3F51B5;
}
li[data-food='burger'] {
    background: #bb6666;
}
```



7.- READING Data Attributes

- The jQuery data() method access data attributes of an element.
- jQuery uses the camelCase version of data attributes.
- jQuery tries to convert the string obtained from a data attribute into a suitable type (number, boolean, object, array, null…). If we don't want that, we should use jQuery's attr() method.

```
<div data-role="page" data-last-value="43" data-hidden="true" data-options='{"name":"John"}'></div>
```

```
$("div").data("role") === "page";
$("div").data("lastValue") === 43; //converted to int and without -
$("div").data("hidden") === true;  //converted to boolean
$("div").data("options").name === "John";
```

7.- WRITING Data Attributes

- The jQuery data() method can be also used to change the values of the data attributes of an element:

```
$("body").data("foo", 52);
$("body").data("bar", {myType: "test", count: 40});
```

7.- REMOVE Data Attributes

- Finally, the removeData() method removes data previously set with the data() method:

```
$("div").data("message", "Hello World");
console.log("Message is: " + $("div").data("message")); // Message is Hello World

$("div").removeData("message");
console.log("Message is: " + $("div").data("message")); // Message is undefined
```